基于 MD5 算法的文件完整性校验程序

1811413 权威

2021 年 5 月 23 日

目录

1 MD5 算法实现 **3**

1.1 初始向量声明 3

1.2 四种基本运算的定义与循环左移 3

1.3 基本函数 3

1.4 sin 取整函数 4

1.5 md5 加密部分 4

1.5.1 填充消息 4

1.5.2 真正 md5 加密 6

1.5.3 显示数据 8

2 主函数各个功能的实现 **8**

2.1 主函数 main 9

2.2 打印帮助信息 10

2.3 测试程序的正确性 11

2.4 计算所给文件的完整性 11

2.5 通过输入验证文件的完整性 13

2.6 通过 MD5 文件校验文件完整性 15

3 项目结构 **16**

4 遇到的问题 **17**

1 MD5 算法实现

1.1 初始向量声明

```
DWORD A=0x67452301;
DWORD B=0xEFCDAB89;
DWORD C=0x98BADCFE;
DWORD D=0x10325476;
```

图 1: 初始向量

这里需要注意的是数据在大小端编址中的不同排列，intel x86 处理器为小端编址，所以初值的声明采用以上方式。

1.2 四种基本运算的定义与循环左移

四种基本运算组成了之后要使用的基本函数。基本函数和四种基本运算都使用宏定义的方式。

```
/*
算法四个基本函数
*/
#define F(x,y,z) (((x)&(y))|((~x)&(z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))
/*
循环左移
*/
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
```

1.3 基本函数

```
#define FF(a, b, c, d, x, s, ac) {\
(a) += F ((b), (c), (d)) + (x) + ac;\
(a) = ROTATE_LEFT ((a), (s));\
(a) += (b);\
}
#define GG(a, b, c, d, x, s, ac) {\
```

```
(a) += G ((b), (c), (d)) + (x) + ac;\
(a) = ROTATE_LEFT ((a), (s));\
(a) += (b);\
}
#define HH(a, b, c, d, x, s, ac) {\
(a) += H ((b), (c), (d)) + (x) + ac;\
(a) = ROTATE_LEFT ((a), (s));\
(a) += (b);\
}
#define II(a, b, c, d, x, s, ac) {\
(a) += I ((b), (c), (d)) + (x) + ac;\
(a) = ROTATE_LEFT ((a), (s));\
(a) += (b);\
}
```

1.4 sin 取整函数

在 FF、GG、HH、II 函数中存在一个常数 T[i] 表示 4294967296Íabs(sin(i)) 的整数部分，因此需要建立一个取整函数来返回 sin 的整数部分。由于没有对于小数求绝对值的函数，因此也需要建立一个取绝对值的函数。

```
double ABS(double value)
{
    if(value<0)
        return -1*value;
    return value;
}

unsigned int T(int i)
{
    ////计算4294967296Íabs(sin(i))的整数部分
    return (unsigned int)(pow_32*ABS(sin((double)i)));
}
```

图 2: 取整函数

这里的 pow_32 定义在 MD5.h 中，其值为 4294967296

1.5 md5 加密部分

1.5.1 填充消息

在 MD5 算法中, 首先需要对输入消息进行填充, 使其比特长度对 512 求余的结果等于 448。也就是说, 消息的比特长度将被扩展至 N×512+448, 即 N×64+56 个字节, 其中 N 为正整数。

具体的填充方法如下: 在消息的最后填充一位 1 和若干位 0, 直到满足上面的条件时才停止用 0 对信息进行填充。然后在这个结果后面加上一个以 64 位二进制表示的填充前的消息长度。经过这两步的处理, 现在消息比特长度 =N×512+448+64=(N+1) ×512。因为长度恰好是 512 的整数倍, 所以在下一步中可以方便地对消息进行分组运算。

因此，当消息不满足长度 =N*64+56 字节时，需要进行填充，具体的填充方法为:

1 计算出需要填充的字节数计为 B，原消息的长度（字节数）计为 A

2 使用 malloc 分配一块内存作为最终的经过填充的消息,大小为 A+B+8 字节，最后 8 个字节是原消息的二进制长度，即有多少 bit

具体填充函数实现如下:

```
byte* padding(byte*source,byte*pad,int length,int total_len, int *res_len)
{
        /*
        返回一个进行了填充的数组
        source为读取到的字节数组
        length为source的长度
        total_len 是要加密的整个数据的长度
        *res_len  是malloc分配的内存空间的长度
        */
        int current_length=length;
        if(length%64!=56||length<64)
        {
                //首先计算出需要填充的字节数
                int count=0;
                while((length+count)%64!=56)
                        count++;
                *res_len=length+count+8;
                pad=malloc(count+length+8);
                //首先复制数组sourcec
```

```
20                  for(int i=0;i<length;i++)
21                          pad[i]=source[i];
22                  pad[length]=0b10000000;//填充1位1
23                  //其余位填充0
24                  for(int i=1;i<=count-1;i++)
25                          pad[length+i]=0;
26                  //然后在这个结果后面加上一个以 64 位二进制
27                  //表示的填充前的消息长度,必须是完整的!!!
28                  //即整个文件的长度
29                  unsigned long t=total_len*8;
30                  for(int i=0;i<8;i++)
31                  {
32                          pad[i+length+count]=(t>>(i*8))&0xff;
33                  }
34                  return pad;
35          }
36          else
37          {
38                  return source;
39          }
40  }
```

1.5.2 真正 md5 加密

在得到了填充后的消息后整个算法分为如下流程:

1 首先将初始向量的值赋给临时变量 a,b,c,d

2 由于我们读取到的是字节流,而在 FF 等函数中加密时需要用到 32bit 大小的元素,也就是一个双字,所以我们还需要将字节流传换为双字流,这通过一个函数来完成。

3 第 2 步完成后,我们就可以用基本函数进行 4 轮 md5 计算了。

4 4 轮计算完成后,将基本向量与 a,b,c,d 分别相加,再拼接在一起就得到了加密后的值。

代码如下：

首先是字节转换函数，负责将 length 字节的数据转换为双字流：

```
void Decode( byte *input, DWORD *output, size_t length)
{
    for(int i=0 , j=0 ;i<length;i+=4,j++)
    {
        output[j]=input[i]|(((DWORD)input[i+1])<<8)|\
        (((DWORD)input[i+2])<<16)|(((DWORD)input[i+3])<<24);
    }

}
```

图 3: 字节转换函数

接下来是 md5 加密部分，x 即为转化以后的双字流：

```
void md5_block_encode( byte* block)
{
    DWORD a = A, b = B, c = C, d = D;
    DWORD*x=malloc(16*4);
    Decode(block, x, block_size);
    /*
        第一轮
    */
    //printf("first round\n");
    FF (a, b, c, d, x[0], 7, T(1));
    FF (d, a, b, c, x[1],12, T(2));
    FF (c, d, a, b, x[2],17, T(3));
    FF (b, c, d, a, x[3],22, T(4));

    FF (a, b, c, d, x[4], 7, T(5));
```

图 4: 加密

在 4 轮加密完成后需要将得到的值与原来的值相加，从而得到最终结果。

```
    II (b, c, d, a, x[9], 21, T(64));
    free(x);
    //printf("final\n");
    A+=a;
    B+=b;
    C+=c;
    D+=d;
```

图 5: 加密

1.5.3 显示数据

在完成了加密后，还需要将结果正确的显示出来，例如，在加密字符串"a" 时,A 的结果最后为:B975C10C,对比"a" 的加密结果 0cc175b9c0f1b6a831c399e269772661 可以看出 A 在内存中的排列并不是最后的正确结果，因此还要对结果继续进行转换，我们定义 DWORD change_end(DWORD N) 函数来完成上述操作，最后用 sprintf 函数将数值传递给结果字符串。

```
DWORD change_end(DWORD N)
{
        byte str[4];
        DWORD res=0;
        for(int i=0;i<4;i++)
        {
                str[i]=(N>>(i*8));
                res|=((DWORD)str[i])<<((3-i)*8);
        }
        return res;
}
// 使用sprintf
sprintf(res,"%08X%08X%08X%08X",change_end(A),change_end(B)\
,change_end(C),change_end(D));
```

2 主函数各个功能的实现

文件完整性检验是在 main 函数中实现的。应用程序为用户提供了多个选项, 不但可以在命令行下计算文件的 MD5 摘要, 验证文件的完整性, 还可以显示程序的帮助信息和 MD5 算法的测试信息。对于这些功能，我们使用一个结构体 CMD_TABLE 的数组来来实现，结构体 CMD_TABLE 定义如下 (cmdtable.h 中):

```
struct CMD_TABLE{
        char *name;//选项名称
        char *description;//选项的描述
        int (*handler) (char *,char*);//选项对应的函数
};

#define NR_CMD 5//选项个数
```

```
8
9  #ifndef oops
10 #define oops(msg){perror(msg);exit(1);} //打印出错信息
11 #endif
12 static int cmd_h(char*arg1,char*arg2);//选项中的函素
13 static int cmd_t(char*arg1,char*arg2);
14 static int cmd_c(char*arg1,char*arg2);
15 static int cmd_v(char*arg1,char*arg2);
16 static int cmd_f(char*arg1,char*arg2);
17 void cal_file_md5(int fd,char*buf);//计算文件的MD5
```

而对应的数组为（cmdtable.c 中）：

```
struct CMD_TABLE cmd_table [] = {
  { "-h", "--help information",cmd_h },
  { "-t", "--test MD5 application",cmd_t },
  { "-c", "[file path of the file computed] \n\t\t--compute \
  MD5 of the given file",cmd_c},
  { "-v", "[file path of the file validated] \n\t\t--validate\
  the integrality of a given file by manual input MD5 value",cmd_v},
  { "-f", "[file path of the file validated] [file path of the\
   .md5 file] \n\t\t--validate the integrality of a given file by read MD5 value from .md5 file",cmd_f},
};
```

图 6: 数组 cmdtable

我们只需要实现每个选项对应的函数以及在主函数中识别选项并调用相关的函数即可。

2.1 主函数 main

主函数读取从终端中输入的选项与路径，将选项与 cmdtable 中的选项进行比较，如果比较成功，则调用相应的函数进行处理。如果没有匹配，则默认打印帮助信息。

主函数实现如下：

```c
extern struct CMD_TABLE cmd_table[];
int main(int ac,char*av[])
{
    if(ac==1)
    {
        cmd_table[0].handler(NULL,NULL);
        return 1;
    }
    for(int i=0;i<NR_CMD;i++)
    {
        if(strcmp(av[1],cmd_table[i].name)==0)
        {
            return cmd_table[i].handler(av[2],av[3]);
        }
    }
    cmd_table[0].handler(NULL,NULL);
    return 1;
}
```

图 7: 主函数部分

2.2 打印帮助信息

打印帮助信息很简单，遍历 cmdtable 输出其中的 name 和 description 即可。

```c
static int cmd_h(char *arg1,char*arg2)
{
    printf("USAGE:\n");
    for(int i=0;i<NR_CMD;i++)
    {
        printf("\t%s %s\n",cmd_table[i].name,cmd_table[i].description);
    }
    return 0;
}
```

图 8: 帮助信息

效果如下:

```
$ ./main.exe h
USAGE:
        -h --help information
        -t --test MD5 application
        -c [file path of the file computed]
                --compute   MD5 of the given file
        -v [file path of the file validated]
                --validate  the integrality of a given file by manual input MD5 value
        -f [file path of the file validated] [file path of the   .md5 file]
                --validate the integrality of a given file by read MD5 value from .md5 file
```

图 9: 帮助信息

2.3 测试程序的正确性

-t 选项用于测试程序的正确性，我们只需要定义一个需要进行测试的字符串数组，然后分别加密即可

```
static int cmd_t(char*arg1,char*arg2)
{
    char res[32];//结果数组
    char *a[]={
        "",
        "a",
        "abc",
        "message digest",
        "abcdefghijklmnopqrstuvwxyz",
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu
        "12345678901234567890123456789012345678901234567
    };
    byte*pad=NULL;
    int len;
    for(int i=0;i<7;i++)
    {
        pad=padding(a[i],pad,strlen(a[i]),strlen(a[i]),&
        for(int j=0;j<len/64;j++)
        {
            md5_block_encode(pad+j*64);
        }
        sprintf(res,"%08X%08X%08X%08X",change_end(A),cha
        printf("\tMD5(\"%s\")=%s\n",a[i],res);
        printf("%08X\n",A);
        free(pad);
        A=A0;//A0保存了A最初的值，其他的也一样
        B=B0;
        C=C0;
        D=D0;
    }
    return 0;
```

图 10: cmd_t 函数

效果如下：

```
./main.exe -t
        MD5("")=D41D8CD98F00B204E9800998ECF8427E
        MD5("a")=0CC175B9C0F1B6A831C399E269772661
        MD5("abc")=900150983CD24FB0D6963F7D28E17F72
        MD5("message digest")=F96B697D7CB7938D525A2F31AAF161D0
        MD5("abcdefghijklmnopqrstuvwxyz")=C3FCD3D76192E4007DFB496CCA67E13B
        MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
        MD5("12345678901234567890123456789012345678901234567890123456789012
```

图 11: cmd_t 函数效果

2.4 计算所给文件的完整性

我们先定义一个函数 cal_file_md5(int fd,char*res) 来对文件计算 MD5,fd 是打开的文件的文件描述符，res 是存放 MD5 结果的数组。

```
void cal_file_md5(int fd,char*res)
{
    byte buf[64];
    int count ;
    //char res[32];
    byte*pad=NULL;
    int len;
    int total_count=0;
    while((count = read(fd,buf,block_size))!=0)
    {
        total_count+=count;
        if(count==block_size)
        {
            md5_block_encode(buf);
        }
        else
        {
            //最后要补的长度是整个数据的长度
            pad=padding(buf,pad,count-1,total_count-1,&len);
            md5_block_encode(pad);
            free(pad);
        }
    }
    sprintf(res,"%08X%08X%08X%08X",change_end(A),change_end(B)
```

图 12: 计算给定文件描述符的 md5

我们以 64 字节为单位进行数据的读取，这里分为两种情况，如果读取的字节数刚好够 64 字节，即文件正好是 64 字节的整数倍或是还没有读完，则直接进行 MD5 加密，如果所读取的字节数不足 64 字节，即文件本身小于 64 字节或是读取了最后不到 64 字节的数据，则需要对数据进行填充，并在最后加上整个文件的二进制长度。

假设当前文件夹下有一个文件 test，里面内容为：12345678901234567890123456789012345678901234567890123456789012345678901234 5678901234567890

先使用 cmd_c 函数打开文件 test，再调用 cal_file_md5 函数进行计算，之后打印计算结果即可。

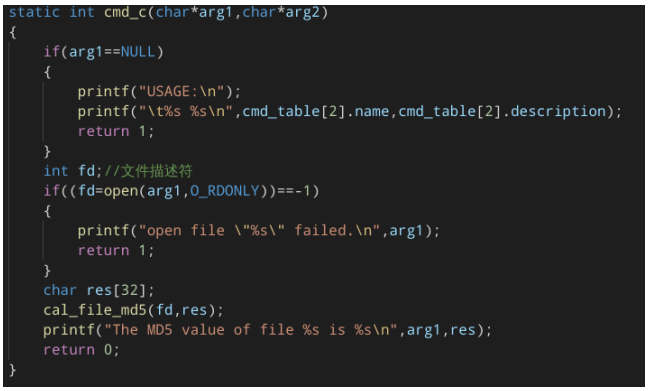

图 13: cmd_c 函数

MD5 计算的效果为:

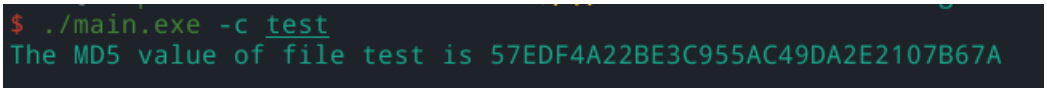

图 14: cmd_c 函数效果

2.5 通过输入验证文件的完整性

我们只需要在前面计算文件 md5 的基础上增加一个字符串校验即可。代码如下:

```
1  static int cmd_v(char*arg1,char*arg2)
2  {
3      if(arg1==NULL)
4      {
5          printf("USAGE:\n");
6          printf("\t%s␣%s\n",cmd_table[2].name,cmd_table[2].description);
7          return 1;
8      }
9      int fd;//文件描述符
10     if((fd=open(arg1,O_RDONLY))==-1)
11     {
12         printf("open␣file␣\"%s\"␣failed.\n",arg1);
```

```
13          return 1;
14      }
15      //读取输入的md5字符串
16      byte read_arr[33];
17      read_arr[32]='\0'
18      printf("input MD5 value of file (\"%s\"):\n",arg1);
19      while(read(0,read_arr,32)!=32)
20      {
21          printf("You must input correct MD5 format string.\n");
22          printf("input MD5 value of file (\"%s\"):\n",arg1);
23      }
24      char res[33];
25      char res[32]='\0'
26      cal_file_md5(fd,res);
27      for(int i=0;i<32;i++)
28      {
29          if(res[i]!=read_arr[i])
30          {
31              printf("Match error! The file has been modified!\n");
32              printf("input MD5:\t\t%s\n",read_arr);
33              printf("calculated file MD5:\t%s\n",res);
34              return 1;
35          }
36      }
37      printf("The file \"%s\" is integrated\n",arg1);
38      return 0;
39  }
```

```
$ ./main.exe -v test
input MD5 value of file ("test"):
57EDF4A22BE3C955AC49DA2E2107B67A
The file "test" is integrated
```

图 15: cmd_v 函数效果

可以看出，此时是正确的。当我们改变输入的 MD5value 时：

```
$ ./main.exe -v test
input MD5 value of file ("test"):
57EDF4A22BE3C955AC49DA2E2107B67B
Match error! The file has been modified!
input MD5:              57EDF4A22BE3C955AC49DA2E2107B67B
calculated file MD5:    57EDF4A22BE3C955AC49DA2E2107B67A
```

图 16: cmd_v 函数效果

会提示文件完整性遭到了破坏。

2.6 通过 MD5 文件校验文件完整性

与前一个功能类似:

```
static int cmd_f(char*arg1,char*arg2)
{
    if(arg1==NULL||arg2==NULL)
    {
        printf("USAGE:\n");
        printf("\t%s %s\n",cmd_table[4].name,cmd_table[4].description);
        return 1;
    }
    int fd_test;//被测文件
    int fd_MD5;//MD5文件
    if((fd_test=open(arg1,O_RDONLY))==-1)
    {
        printf("open file \"%s\" failed.\n",arg1);
        return 1;
    }
    if((fd_MD5=open(arg2,O_RDONLY))==-1)
    {
        printf("open file \"%s\" failed.\n",arg1);
        return 1;
    }
    //对测试文件计算md5
    char res_test[33];
    res_test[32]='\0';
    cal_file_md5(fd_test,res_test);
    //读取md5 file中的值
    char res_md5[33];
    res_test[32]='\0';
```

```
28      if(read(fd_MD5,res_md5,32)!=32)
29      {
30          printf("read file \"%s\" failed\n",arg2);
31          return 1;
32      }
33      for (int i = 0; i < 32; i++)
34      {
35          if(res_md5[i]!=res_test[i])
36          {
37              printf("Match error! The file has been modified!\n");
38              printf("MD5 file value:\t\t%s\n",res_md5);
39              printf("calculated file MD5:\t%s\n",res_test);
40              return 1;
41          }
42      }
43      printf("The file \"%s\" is integrated\n",arg1);
44      return 0;
45  }
```

3 项目结构

主要分为 MD5 加密部分和从命令行解析选项并执行这两大块，并分别建立了.h 和.c 文件。

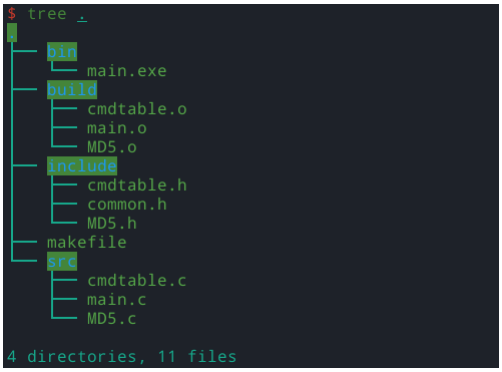

图 17: 项目结构

common.h 主要定义了数据类型 byte 和 DOWRD。

4 遇到的问题

主要是关于大小端的问题，把数据排列方式转化一下就可以解决了。

另外 C 中并没有数组越界的提醒，所以对于字符串数组，需要多一位来存放数组的结束符号, 否则会产生一些奇奇怪怪的错误。